VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Lauri Valkola

CAN-LAITTEIDEN TESTAUSSOVELLUS

Tekniikka ja liikenne
2009

ALKUSANAT

Tämä opinnäytetyö tehtiin Vaasan ammattikorkeakoulun tietotekniikan koulutusohjelmassa Wapice Oy:n toimeksi antamana. Wapice Oy:n yhdyshenkilönä toimi projektipäällikkö Jari Kuusisto ja Vaasan ammattikorkeakoulun valvovana opettajana filosofian tohtori Ghodrat Moghadampour.

Haluan kiittää työtoveriani insinööri Jari Kuusistoa arvokkaasta avustaan, ja filosofian tohtori Ghodrat Moghadampouria asiantuntemuksestaan. Kiitokset myös Wapice Oy:lle opinnäytetyön aiheesta.

Vaasassa 13.12.2009

Lauri Valkola

VAASAN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Lauri Valkola
Opinnäytetyön nimi	CAN-laitteiden testaussovellus
Vuosi	2009
Kieli	Suomi
Sivumäärä	55
Ohjaaja	Ghodrat Moghadampour

Opinnäytetyön tarkoituksena oli luoda ohjelmisto jolla pystytään tarkastelemaan CAN (Controller-Area Network)-väylän liikennettä sekä lähettämään viestejä väylälle. CAN on yksi johtavista sarjaliikennejärjestelmistä. Vaatimuksiin kuului viestien lähettämisen ja lukemisen lisäksi myös seuraavat toiminnot: kuorman luominen väylälle, skriptituki, reititys CAN-laitteelta toiselle, triggerien konfigurointi sekä viestien nauhoittaminen ja toistaminen. Sovelluksen haluttiin toimivan PC:llä Windows-ympäristössä PC:hen kytkettävien CAN-sovittimen avulla. Sovelluksen tuli myös olla helposti kehitettävissä jatkossa sekä laajennettavissa Linux-pohjaisiin järjestelmiin. Lisäksi ohjelman eri komponentteja tuli voida käyttää tulevaisuuden projekteissa.

Käyttöliittymä toteutettiin käyttäen Qt-käyttöliittymäkirjastoa. Qt valittiin koska se tukee eri alustoja, kuten Windows, Linux, Mac OS sekä sulautetut alustat. Qt valittiin myös koska se oli työpaikan aiemmissa projekteissa todettu toimivaksi ja helppokäyttöiseksi. Käyttöliittymä toteutettiin työpöytämalliseksi.

Toteutetusta ovelluksesta löytyy toiminnot väylälle kirjoittamiseen ja väylältä lukemiseen. Näitä toimintoja voi käyttää monella laitteella samanaikaisesti. Sovelluksessa toteutettiin myös alatason CAN-kirjasto. Alkuperäisestä suunnitelmasta poiketen jäivät muut toiminnot toteuttamatta annetun aikataulun puitteissa.

Asiasanat C++, Qt, CAN, Testaus

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tietotekniikan koulutusohjelma

ABSTRACT

Author	Lauri Valkola
Title	Test application for CAN-devices
Year	2009
Language	Finnish
Pages	55
Name of Supervisor	Ghodrat Moghadampour

The purpose of this thesis work was to develop a software that helps us to observe the traffic on a CAN (Controller-Area Network)-bus, and send messages to the bus. CAN is one of the leading serial bus systems. In addition to reading and writing to the bus the requirements presuppose busload generating, script support, routing from one device to another, trigger configuration, message recording and playing the recordings. The software was created to run on a PC in a Windows-environment with CAN adapters that are plugged to the PC. The requirements included that the software has to be easy to develop further and it has to be portable to Linux-based systems.

The user interface was developed using Qt, which is a cross-platform and UI Framework. Qt was chosen for this project because it supports different platforms such as Windows, Linux, Mac OS and embedded platforms. Another reason was that it had been proven functional and easy to use by employees at the company that requested this software.

The final application has functionalities for reading from the bus and writing to the bus. These functionalities can be used with many devices simultaneously. A generic low level CAN-library was developed as a part of the software. Differing from the original plan, the rest of the functionalities were not implemented within the scope of the work.

Keywords C++, Qt, CAN, Testing

MERKINNÄT JA LYHENTEET

API	*Application Programming Interface.* Ohjelmointirajapinta
Baudinopeus	Tarkoittaa määrää mahdollisia tilanvaihdoksia kommunikaatiokanavassa sekunnissa
CAN	*Controller-Area Network.* Teollisuudessa yleisessä käytössä oleva sarjaliikennekommunikointijärjestelmä
Eclipse	Avoimen lähdekoodin lisensillä toimiva kehitysympäristö
GCC	*Gnu Compiler Collection.* Avoimen lähdekoodin lisensillä toimiva kääntäjäkokoelma
GNU	GNU's Not Unix. Järjestö omistautunut vapaan lähdekoodin ohjelmistoihin
Instanssi	Luokasta alustettu olio, eli luokan edustaja
Kirjasto	Kokoelma funktioita ja luokkia
PCMCIA	*Personal Computer Memory Card International Association.* Yleensä kannettavista tietokoneista löytyvän laajennuskorttipaikan tyyppi
Puskuri	Ohjelmoinnissä käytetty termi muistinvarauksesta, jonka läpi tieto kulkee ja jossa se voi odottaa käsittelyä
Qt	Nokian omistamaa kehitysympäristö ja käyttöliittymäkirjasto
USB	*Universal Serial Bus.* Yleinen sarjaväylä oheislaitteiden liittämiseksi tietokoneeseen

Sisällys

TIIVISTELMÄ....3
ABSTRACT....4
MERKINNÄT JA LYHENTEET....5
1 JOHDANTO....8
1.1 Wapice Oy....8
1.2 Työn kuvaus....8
1.3 Historia....9
1.4 CAN-laitteet....9
1.5 Ominaisuuksia....12
1.6 CAN-viestien rakenne....13
1.6.1 Datakehys....13
1.6.2 Etäkehys(Remote frame)....15
1.6.3 Virhekehys....16
1.6.4 Ylikuormituskehys....17
2 OHJELMISTON MÄÄRITTELY....18
2.1 Työn määrittely....18
2.2 Tavoitteet....18
2.3 Vaatimusmäärittely....18
2.4 Ohjelman käyttötapaukset....21
2.4.1 Luo uusi työpöytä....22
2.4.2 Avaa tallennettu työpöytä....22
2.4.3 Tallenna työpöytä....23
2.4.4 Näytä yhdistetyt laitteet....23
2.4.5 Yhdistä väylälle....23
2.4.6 Sulje yhteys....24
2.4.7 Kirjoita viesti väylälle....25
2.4.8 Lue viestejä väylältä....26

2.5 Sovelluksen luokat....27
2.5.1 CANHandlerAPI....28
2.5.2 DeviceData ja DeviceDataInfo....31
2.6 Sovelluksen rakenne....36
2.7 Arkkitehtuuri....37
3 KÄYTTÖLIITTYMÄN SUUNNITTELU....39
3.1 Diagnostiikkaikkuna....39
3.2 Viestin lähetysikkuna....40
3.3 Kuormitusikkuna....40
4 TOTEUTUS....42
4.1.Työkalut....42
4.1.1 Tarvittavat työkalut....42
4.1.2 Kehitysympäristö....42
4.1.3 Kääntäjä....43
4.1.4 Ulkoiset laitteet....43
4.2 Teknologia....43
4.2.1 Toteutuskieli....43
4.2.2 Käyttöliittymä....44
4.2.3 Laiteläheinen ohjelmointi....44
4.3 Viestin luku IXXAT-laitteelta....44
4.4 Viestin luku CANHandlerAPI-luokassa....46
4.5 Viestin kirjoittaminen väylälle....49
4.6 Ikkunoiden luominen käyttöliittymässä....50
5 YHTEENVETO....52
5.1 Toteutus....52
5.2 Jatkokehittelyn näkymät....52
LÄHDELUETTELO....53

1 JOHDANTO

Ympäröivässä maailmassa on monia elektroniikkaan perustuvia laitteita, ja näiden laitteiden elektroniset osat voivat olla fyysisesti erillään. Näiden laitteiden on kuitenkin voitava kommunikoida keskenään ja tässä syntyvien mahdollisten ongelmakohtien vuoksi on myös tärkeätä pystyä testaamaan näitä laitteita niiden kommunikointia. Tämän vuoksi ryhdyttiin tässä työssä kehittämään ohjelmaa jolla näitä laitteita voi testata. Työ toteutettiin Wapice Oy:n toimesta.

1.1 Wapice Oy

Wapice Oy on vuonna 1999 Vaasassa perustettu yritys. Yritys on henkilöstön omistama, ja nykypäivänä toimipisteitä on Vaasan lisäksi Tampereella, Oulussa, Hyvinkäällä ja Seinöjoella.

Wapice Oy toteuttaa asiakkaiden vaatimusten mukaan projekteja elektroniikkaasuunnittelun, ohjelmistosuunnittelun ja ohjelmistotototeutuksen alueilla. Useimmat yrityksen asiakkaista ovat Suomen 200 suurimman teollisuusyrityksen joukossa. Wapice Oy on kehittänyt myös useita teollisuusvaatimukset täyttäviä tuotteita, liittyen muun muassa etähallintajärjestelmiin, liiketoimintaohjelmistoihin sekä sulautettuihin laitteisiin.

1.2 Työn kuvaus

Tässä dokumentissa perehdytään CAN-järjestelmään sekä tämän viestirakenteeseen. Sitten työssä käydään läpi toteutettavan sovelluksen vaatimusten määrittely, toiminnallisuuksien selvitys komponenttitasolla, sekä näiden toteutustavat. Esitetään minkälaisiin komponentteihin ohjelma jaetaan ja miten ohjelmiston loogiset osat toteutettiin erikseen. Näytetään ohjelmiston eri käyttötapauksia sekä näihin perustuvia toimintaketjuja ohjelmiston eri komponenttien välillä.

Työn lopputuloksena saatiin testityökalu jolla voidaan kirjoittaa CAN-väylälle ja lukea väylältä. Sovellus tarjoaa myös hyvän pohjan lisäominaisuuksien

kehittämiselle. Osana sovellusta toteutettiin geneerinen alatason CAN-kirjasto joka on helposti otettavissa käyttöön tulevissa projekteissa .

1.3 Historia

CAN on sarjaliikennejärjestelmä joka kehitettiin 1980-luvun alussa autoteollisuuden tarpeisiin. Ensimmäisen kerran sarjaliikennejärjestelmä esiteltiin helmikuussa 1986 Society of Automotive Engineersin tapaamisessa Robert Boschin toimesta. Boschin tehdas tarkkaili 1980-luvulla olemassaolevia sarjaliikenneväyliä, mutta yksikään näistä ei täyttänyt heidän asettamiaan vaatimuksia, ja näin he aloittivat uuden sarjaliikennejärjestelmän kehittämisen. Kehitysvaiheessa liittyi kehittämiseen Mercedes-Benzin insinöörejä, Intelin työtekijöitä ja professori Dr. Wolfhard Lawrenz sovellettujen tieteiden yliopistosta Braunschweig-Wolfenbüttelista Saksasta palkattiin konsultiksi. Hän myös antoi verkolle nimen. |3|

CANista syntyi yksi menestyksekkäimmistä verkkoprotokollista ja voi sanoa, että ainakin yksi CAN-verkko löytyy lähes jokaisesta Eurooppassa valmistetusta autosta. Nykyään CAN-väylää käytetään ajoneuvoissa ja rakennuskoneissa, sekä myös teollisuustuotannossa, maataloudessa, terveydenhuollossa ja monella muulla alalla. CAN on tänä päivänä maailmanlaajuisesti yksi suosituimmista väyläprotokollista. |3|

1.4 CAN-laitteet

CAN-laite on mikroprosessorilla varustettu elektrooninen laite joka pystyy ottamaan vastaan ja/tai lähettämään CAN-viestejä. CAN-laitteiden tehtäviä voi olla virheistä ilmoittaminen, mekaanisten osien ohjaaminen, eri laitteiden pois/päälle kytkeminen tai tietojen näyttäminen käyttäjälle. Järjestelmässä laitteiden on esimerkiksi pystyttävä kysymään toisiltaan tilatietoja voidakseen toimia tämän perusteella, tai lähetettämään toisilleen komentoja virhetilojen ja sensoriarvojen perusteella. Alla olevassa taulukossa on esitetty CANopen laite-profiileille kehitetty standardi, josta näkee miten laitteet luokitellaan. CANopen

on suosittu CANiin perustuva järjestelmä.

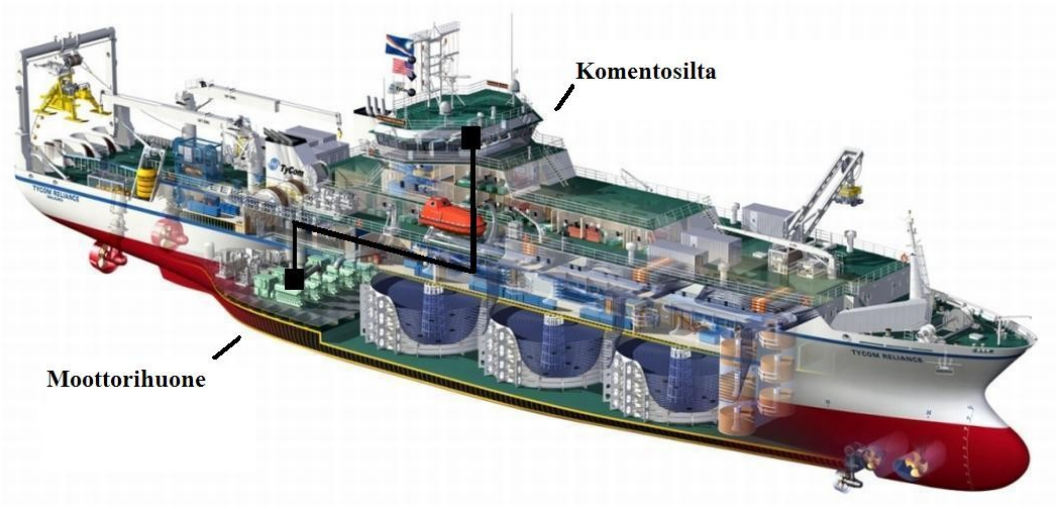
Taulukko 1. CANopen laiteprofiilit |2|

Profiili nro	**Laitteen luokka**
CiA 401	Geneerinen input/output moduuli
CiA 402	Sähkölähteet ja Liikehallintaa
CiA 404	Mittauslaitteet ja suljetun silmukan ohjain
CiA 405	IEC 61131-3 ohjelmoitavat laitteet
CiA 406	Pyörivät ja lineaariset enkooderit
CiA 408	Hydrauliset ohjaimet ja venttiilit
CiA 410	Kaltevuusmittarit
CiA 412	Lääketieteen laitteet
CiA 413	Kuorma-auton yhdyskäytävät
CiA 414	Langansyöttölaitteet
CiA 415	Tientyöstölaitteet
CiA 416	Rakennuksen ovien hallinnointi
CiA 417	Nosturin hallintajärjestelmät
CiA 418	Patterimoduulit
CiA 419	Patterilaturit
CiA 420	Puserrinlaitteet
CiA 422	Kunnalliskulkuneuvot
CiA 423	Rautatien dieselhallintajärjestelmät

CiA 424	Raidekulkuneuvojen ovienhallintajärjestelmät
CiA 425	Medical Diagnostic Add-on Modules
CiA 445	RFID-Laitteet

CAN on järjestelmä joka soveltuu älykkäiden laitteiden, sensoreiden ja käyttölaitteiden verkoittamiseen. CAN-järjestelmiä on käytössä monenlaisissa laitteissa, kuten esimerkiksi erilaisissa kulkuneuvoissa, rakennusalalla, ja teollisuuden laitteissa. Laitteiden verkoittamista tarvitaan koska näiden on pystyttävä kommunikoimaan keskenään jotta koko järjestelmä toimisi tarkoituksenmukaisesti. Esimerkiksi laivassa voi olla monia CAN-laitteita jotka ohjaavat moottoreita, ilmastointia, lämmitystä ja muita toimintoja.

Käyttötilanteessa CAN-järjestelmä laivassa voi toimia seuraavanlaisesti. Laivassa komentosillalla on tietokone, joka käyttää CAN-laitetta kommunikoidakseen laivan kaiken laitteston kanssa. Laivassa on CAN-laitteiden joukossa konehuoneessa ohjain laivan moottoreille. Laiva on lähdössä satamasta ja kapteeni antaa käynnistyskomennon moottoreille. CAN-viesti lähtee komentosillalta väylälle ja moottoreiden CAN-laite huomaa että viesti on tarkoitettu sille. Moottoreiden ohjain lähettää väylälle kuittauksen otettuaan viestin vastaan. Komentosillan CAN-laite saa kuittauksesta tiedon, että viestin lähetys onnistui. Moottorien ohjain yrittää käynnistää moottorit, mutta yksi moottoreista vioittuu eikä käynnisty ollenkaan. CAN-laite huomaa vian ja lähettää väylälle viestin joka kertoo virhetilanteesta. Komentosillan CAN-laite kuittaa viestin ja kapteenin näytölle ilmestyy virheviesti. Kapteeni ryhtyy jatkotoimenpiteisiin. (Kuva 1) |6|



Kuva 1. CAN-järjestelmä laivassa

1.5 Ominaisuuksia

CAN-järjestelmässä on hyviä ominaisuuksia jotka ovat auttaneet sen maailmanlaajuiseen menestykseen. Tärkeimpiä näistä ovat:

- Usean isännän valmius. Tämä tarkoittaa sitä että järjestelmä ei ole riippuvainen yhdestä isännästä, vaan kaikki solmut voivat lähettää viestejä väylälle jos väylä on vapaa ja täten toimia tilapäisenä isäntänä.

- Yleiset lähetykset. Kaikki solmut, jotka ovat kiinni väylällä saavat viestit. Solmut päättävät itse haluavatko vastaanottaa viestin. Tämä luo yhtenäisyysvarmuutta järjestelmän tietoon, sillä kaikki solmut käyttävät samaa tietoa.

- Kehittyneitä virheentunnistamismekanismeja, ja virheellisten viestien uudelleenähtetys. Tämä tehostaa järjestelmän varmuutta.

- Viestien priorisointi. Jos kaksi solmua lähettää viestejä samanaikaisesti taataan että suuremmalla prioriteetillä oleva viesti saadaan lähettää ensimmäiseksi.

- Fyysiset vaatimukset. CAN järjestelmä tarvitsee vain kaksi johdinta toimiakseen. |3|

CAN-protokollassa eivät monen muun protokollan tapaan vastaanottajatiedot ole missään kehyksessä, vaan viestin tunnistekenttä kertoo viestin prioriteetin ja minkälaista sisältöä viestissä on. Vastaanottavan laitteen on itse päätettävä otetaanko viesti vastaan.

Monia CAN-protokollia on kehitetty alkuperäisen standardin päälle. Joitakin yleisimpiä näistä ovat CANOpen, DeviceNet ja J1939.

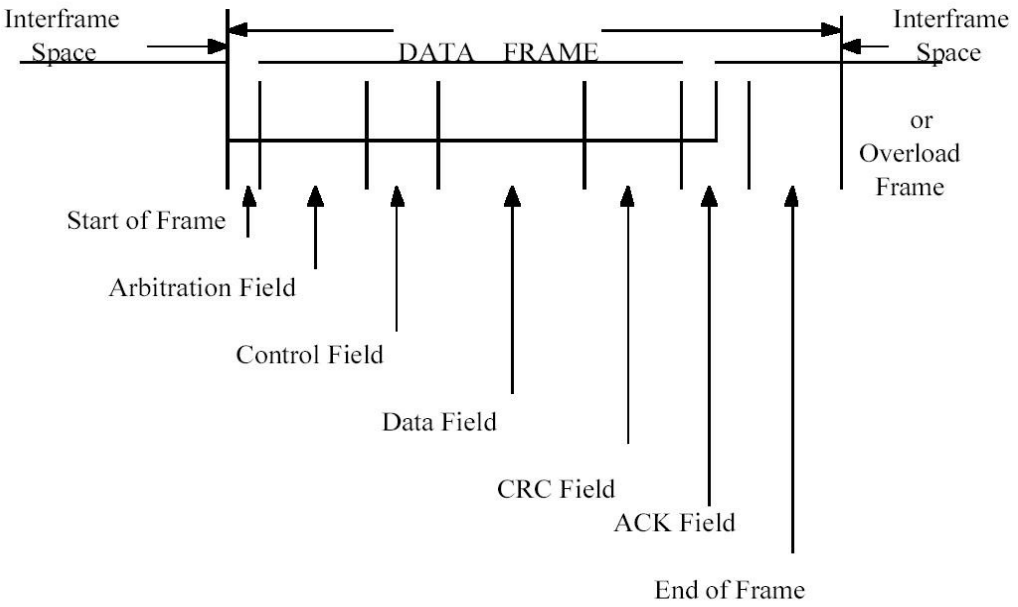
1.6 CAN-viestien rakenne

CAN protokolla tukee kahta eri viestiformaattia: standard- ja uudempi extended formaatti. Standardformaatissa tunnistekenttä on 11-bittinen kun taas extended formaatissa se on 29-bittinen. Kaikkien CAN-laitteiden on tuettava standard-formaattia ja siedettävä extended-formaattia.

CAN-viestejä on neljä erilaista: Datakehys (Data frame), Etäkehys (Remote frame), Virhekehys (Error frame) ja Ylikuormituskehys (Overload frame). |4|

1.6.1 Datakehys

Datakehys on datansiirtoa varten lähettäjältä vastaanottajalle. Alla olevassa kuvassa näkyy kehyksen rakenne (Kuva 2). Datakehys koostuu seuraavista kentistä:



Kuva 2. Datakehyksen rakenne

Kehyksen alku (Start of frame)

Kehyksen alku sijaitsee datakehyksen lisäksi myös etäkehyksessä. Kehys koostuu vain yhdestä bitistä, joka merkitsee kehyksen alkua. Viestin lähettäjä vetää tässä väylän alas alempaan jännitepotentiaaliin. Tämä toiminto mahdollistaa sen, että muut laitteet voivat synkronoida itsensä tulevaan viestiin.|4|

Sovittelukenttä (Arbitration field)

Sovittelukenttä koostuu tunnisteesta (identifier) sekä rtr-bitistä (rtr-bit). Tunnisteen pituus on 11-bittiä. Tunnisteen bitit lähetetään eniten merkitsevä bitti ensiksi (MSB). Rtr-bitti on alemman potentiaalin bitti. Rtr-bitti päättää sovittelukentän. |4|

Hallintakenttä (Control field)

Hallintakenttä koostuu kuudesta bitistä. Neljä bittiä ovat datan pituustunniste(data length code), ja loput kaksi on varattu tulevaisuden laajennuksia varten. Nämä kaksi bittiä ovat aina ylemmällä jännitasolla. Datan pituustunniste kertoo kuinka

monesta tavusta datakenttä(data field) koostuu. Tämä arvo voi olla nollasta kahdeksaan. Tästä kentästä lähetetään ensin kaksi varattua bittiä. Tämän jälkeen lähetettään neljä databittiä eniten merkitsevä bitti ensiksi. |4|

Datakenttä (Data Field)

Datakenttä koostuu viestin datasta. Data voi olla nollasta kahdeksaan tavua pitkä.

CRC-kenttä (cyclic redundance code)

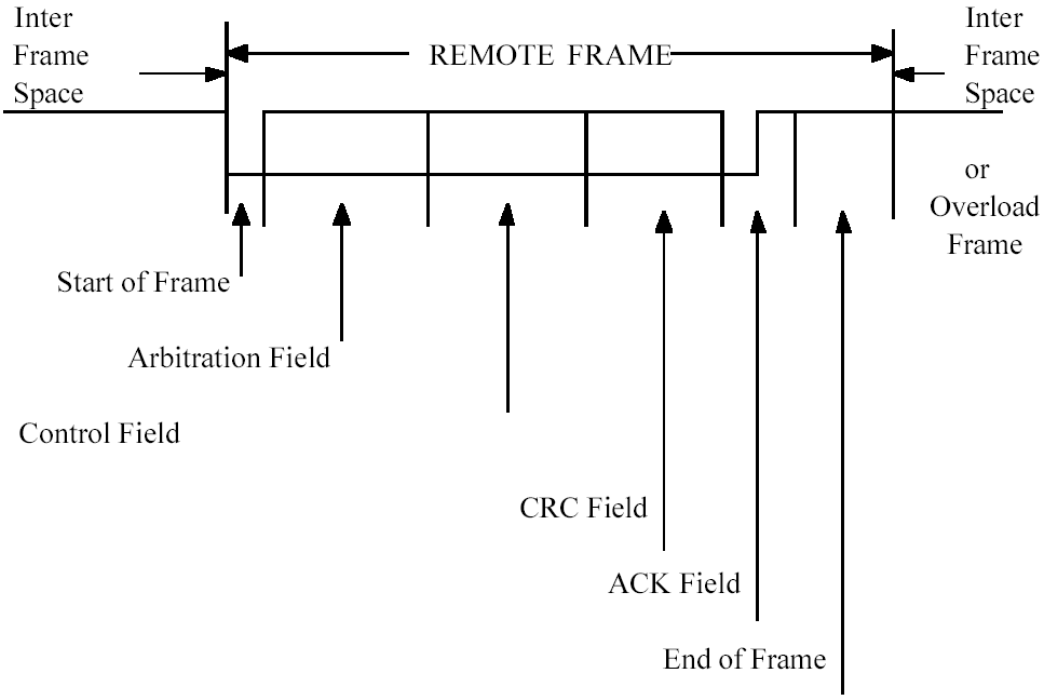
CRC-kenttä on virheentarkistusta varten. Edellä olevista kentistä lasketaan tietyn kaavan mukaan tämän kentän arvo, josta vastaanottaja voi tarkistaa, että viesti on vastaanotettu ilman virheitä. CRC-rajaus on välimerkki joka merkitsee CRC-kentän loppumista. Tämä bitti on yksi alemman jännitetason bitti. |4|

Kuittauskenttä

Kuittauskenttä on kaksi bittiä pitkä. Tämä koostuu lähettäjän lähettämästä bitistä, joka on alemmalla jännitasolla ja vastaanottajien kuittauksesta eli ylemmän tason bitistä. Tämän toiminnon avulla lähettäjä saa tiedon, että viesti on toimitettu oikein. Loppukehys koostuu seitsemästä alemman tason bitistä, jotka merkitsevät viestin loppumista. Samanlainen kehys on myös etäkehyksessä. (Remote frame)|4|

1.6.2 Etäkehys(Remote frame)

Etäkehys on viesti jonka, jonkin tietyn datan vastaanottajana toimiva yksikkö voi lähettää. Tämän viesti on pyyntö lähettää dataa, ja tämän viestin vastaanottajan on vastaanoton jälkeen tarkoitus lähettää pyydetty viesti. Kuvassa 3 näkyy viestin rakenne. |4|

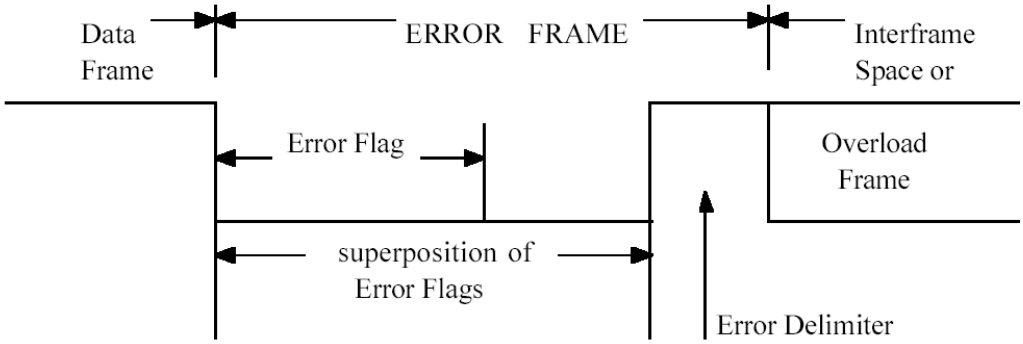


Kuva 3. Etäkehyksen rakenne

Etäkehys eroaa datakehyksestä vain kahdessa suhteessa: Etäkehyksen rtr-bitti on alemmassa jännitetasossa, kun datakehyksessä rtr-bitti oli ylemmässä jännitesossa. Etäkehyksessä ei myöskään ole datakehystä. |4|

1.6.3 Virhekehys

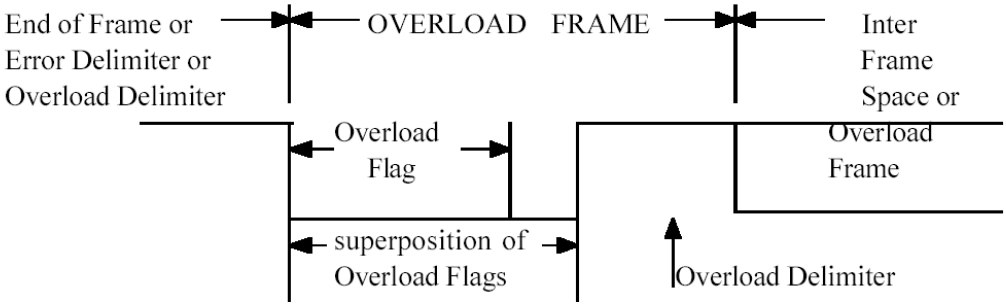
Virhekehys koostuu kahdesta eri kentästä, virhelippujen alueesta ja virhe-erotinkentästä. Kun tapahtuu virhe laite lähettää virhelipun, jonka rakenne rikkoo muiden viestityyppien alun rakennetta. Tästä seuraa, että nämäkin laitteet lähettävät virheliput, joskaan ei täsmälleen samaan aikaan kun ensimmäinen laite. Nämä lähetykset sijoittuvat tähän ensimmäiseen alueeseen. Virhe-erotinkenttä koostuu kahdeksasta alemman tason bitistä. Virhelippuja on kahdenlaisia, aktiivinen virhelippu, joka koostuu kuudesta ylemmän tason bitistä ja passiivinen virhelippu, joka koostuu kuudesta alemman tason bitistä. Laite joka huomaa virheen lähettää aktiivisen virhelipun ja muut laitteet lähettävät passiivisia virhelippuja. Kuvassa 4 näkyy virhekehyksen rakenne. |4|



Kuva 4. Virhekehyksen rakenne

1.6.4 Ylikuormituskehys

Ylikuormituskehys koostuu kahdesta kentästä, ylikuormituslipusta ja ylikuormituserottajasta. Ylikuormituslippu koostuu kuudesta ylemmän tason bitistä ja ylikuormituserottaja koostuu kahdeksasta alemman tason bitistä. Ylikuormitustilanne voi johtua kahdesta syystä: Jos bittejä havaitaan väylällä liian nopeasti edellisen viestin loputtua tai jos vastaanottaja sisäisistä syistä vaatii viivettä ennen seuraavan data-, tai etäkehyksen lähettämistä. Kuvassa 5 näkyy ylikuormituskehyksen rakenne. |4|



Kuva 5. Ylikuormituskehyksen rakenne

2 OHJELMISTON MÄÄRITTELY

2.1 Työn määrittely

Tämä dokumentti käsittelee. sovellusta CAN-väylää ja CAN-protokollaa käyttävien laitteiden analysointiin ja testaukseen. Sovelluksen on tarkoitus toimia työkalupakettina CAN-väylän laitteiden ja kokoonpanojen testaukseen, analysointiin, ongelmien etsimiseen ja ratkaisuun. Sovellusta on tehty ajettavaksi PC:llä Windows-käyttöliittymällä ja sovellus on toteutettu C++-ohjelmoinikielellä. Vaatimukset perustuivat pääsääntöisesti yrityksen omiin vaatimuksiin mutta sen tarjoaminen asiakkaille tulevaisudessa ei ole poissuljettua.

2.2 Tavoitteet

Ohjelmiston on tarkoitus toimia monipuolisena testityökaluna CAN-väylän tarpeisiin. Tarkoituksena on että yritys sekä asiakkaat voivat hyödyntää ohjelmistoa testauksessa, vianetsinnässä, analyysissä, ongelmanratkaisemisessa, muissa CAN-väylään liittyvissä tehtävissä. Ohjelmiston käyttö vaatii käyttäjältä jonkin verran tuntemusta käyttämästään laitteistosta ja CAN-viestinnästä yleisesti.

Tavoitteena oli myös tehdä alemman tason rajapinnasta niin monipuolinen ja helppokäyttöinen että sitä voi käyttää yrityksen tulevissa projekteissa. Tähän osaan projektia käytettiin paljon aikaa suunnitteluun, jotta rajapinnasta saataisiin mahdollisimman tehokas. Koska viestejä voi liikkua väylällä tuhansia sekunnissa piti myös tämä osa ohjelmistosta olla tarpeeksi suorituskykyinen, jotta pystytään vastaamaan teollisuuden tarpeisiin.

2.3 Vaatimusmäärittely

Ohjelmistolle asetetut vaatimukset ovat jaoteltuina prioriteettien mukaan taulukoissa. Ensimmäinen taulukko kuvaa tärkeimmät, toinen vähemmän tärkeät, ja kolmas vähiten tärkeät ominaisuudet.

Taulukko 2. Pakolliset (must have) ominaisuudet

Ominaisuus	**Selitys**
Viestien lähetys	Ohjelmalla tulee pystyä lähettämään CAN-viestejä väylälle.
Viestien lukeminen	Ohjelmalla pitää pystyä lukemaan CAN-viestejä väylältä reaaliajassa.
Geneerinen alatason CAN-kirjasto	Ohjelmasta riippumaton kirjasto, jota voi käyttää tulevissa sovelluksissa. Kirjaston toiminnallisuuksien avulla voi toteuttaa kaikki sovelluksen vaatimukset.
Monen työpöydän konfigurointi	Käyttäjän pitää pystyä luomaan monta eri työpöytää ja vaihdella näiden välillä vapaasti.
Monen kanavan tuki	Sovelluksessa pitää pystyä käyttämään monta laitetta samanaikaisesti ja näiden kaikkia kanavia.

Taulukko 3. Toivotut (should have) ominaisuudet

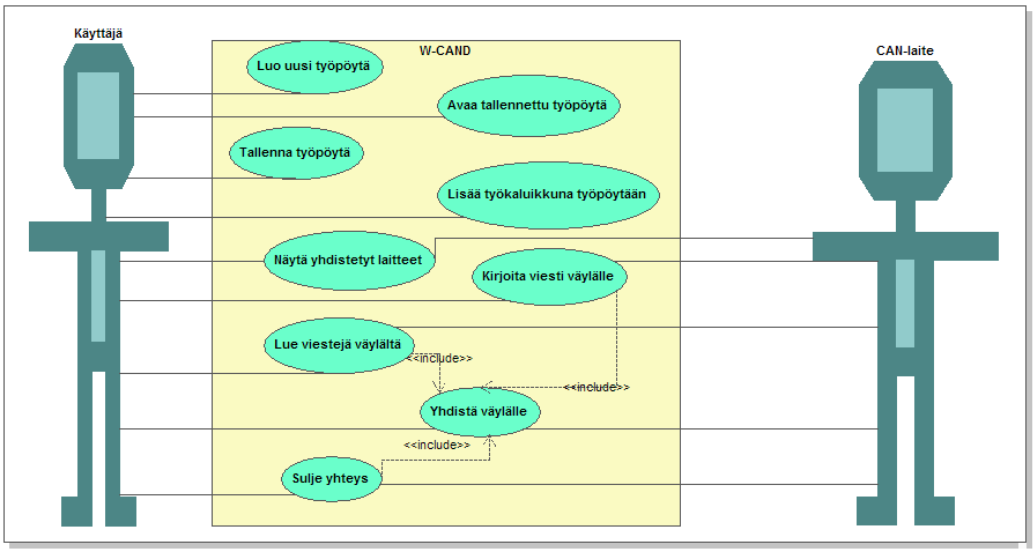
Ominaisuus	**Selitys**
Kuormituksen luominen	Ohjelmassa pitää voida luoda erityyppistä liikennettä väylälle kuormitusmielessä.
Skriptituki	Mahdollisuus tehdä omia skriptejä, jotka suorittavat eri toimintoja.
Linux-tuki	Ohjelmasta tulee muokata versio, joka toimii Linux-käyttöjärjestelmällä.
Reititys	Mikäli monta laitetta on käytössä, voi reitittää yhteen laitteeseen sisääntulevat viestit toisen laitteen lähetettäviin viesteihin.

Taulukko 4. Valinnaiset (nice to have) ominaisuudet

Ominaisuus	**Selitys**
Triggerien konfigurointi	Mahdollisuus määrittää “liipaisimia” jotka käynnistävät esimerkiksi viestin lähetyksen tai viestien nauhoituksen.
Viestien nauhoittaminen	Sisääntulevista viesteistä luotava nauhoite, jossa on kaikki viestien tiedot tarkkaa aikaleimaa myöten.
Viestien toistaminen	Yllä määritellyn nauhoitteen tarkka toistaminen aikaleimojen mukaan.
Linux-tuki	Ohjelmasta tulee muokata versio joka toimii Linux-käyttöjärjestelmällä.
Reititys	Mikäli monta laitetta on käytössä, voi reitittää yhteen laitteeseen sisääntulevat viestit toisen laitteen lähetettäviin viesteihin.

2.4 Ohjelman käyttötapaukset

Ohjelma on yhden käyttäjän ohjelma, ja tästä syystä käyttötapaukset ovat yksinkertaisia. Työpöytään liittyvät käyttötapaukset ovat uuden työpöydän luonti, tallennetun työpöydän avaaminen ja työpöydän tallentaminen. Muita yleisiä käyttötapauksia ovat viestien kirjoitus ja luku, väylältä poistuminen ja väylälle meneminen sekä yhdistettyjen laitteiden haku. Viestien lukeminen ja viestien kirjoittaminen ei onnistu ellei laite ole yhdistetty väylälle. Yhdistäminen on siis vaatimuksena näille käyttötapauksille. Yhteyttä ei myöskään voida sulkea ennen kuin laite on ensin yhdistetty väylälle. Kuvassa 6 on esitetty nämä käyttötapaukset.



Kuva 6. Ohjelman toiminnot

Seuraavassa käydään nämä käyttötapaukset yksityiskohtaiset läpi. Alla olevissa kaavioissa on kuvattu suurin osa tapauksista sekä niiden vaiheet ohjelmassa. Koska kaikki työpöytää koskevat operaatiot suoritetaan ohjelmassa käyttöliittymätasolla, ei ole kovin kuvaavaa kertoa näiden käyttötapauksien vaiheista ohjelmatasoisella kaaviolla.

2.4.1 Luo uusi työpöytä

Käyttäjä valitsee workspace-valikosta “Create new workspace”-kohdan, jolloin esiin ilmestyy ruutu nimensyöttöä varten. Käyttäjä kirjoittaa haluamansa nimen ja painaa ok-painiketta. Käyttöliittymä luo uuden työpöydän.

2.4.2 Avaa tallennettu työpöytä

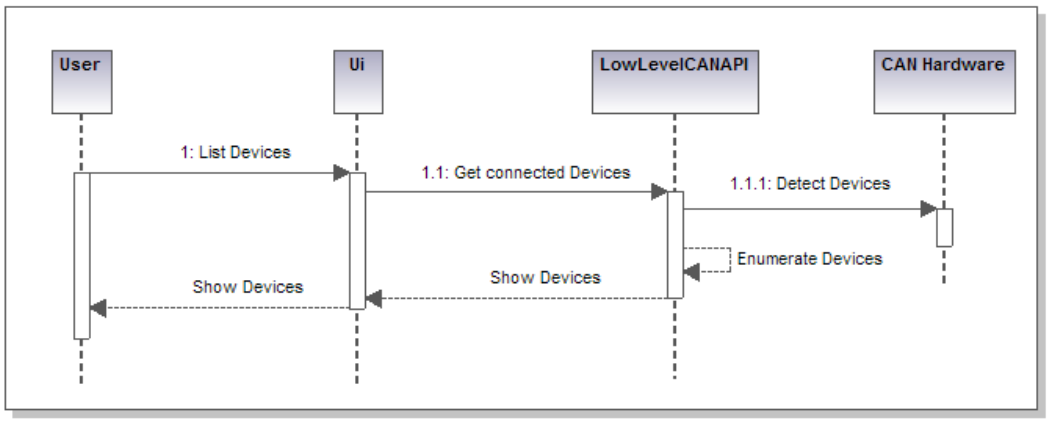
Käyttäjä valitsee workspace-valikosta “Open workspace”-kohdan. Käyttöliittymä avaa tiedostonavaus-dialogin, josta käyttäjä etsii aiemmin tallettamansa työpöydän. Käyttäjä hyväksyy valinnan ja tallennettu työpöytä aukeaa ohjelmaan.

2.4.3 Tallenna työpöytä

Käyttäjä luo itselleen työpöydän lisäten haluttuja ikkunoita. Käyttäjä valitsee workspace-valikosta “Save workspace”-kohdan, jolloin avautuu dialogi tallentamista varten. Kun dialogiin on annettu nimi ja valinta on hyväksytty, tallentuu työpöytä tiedostoon.

2.4.4 Näytä yhdistetyt laitteet

Käyttäjä käskee käyttöliittymän kautta näyttämään yhdistetyt laitteet (1). Käyttöliittymä lähettää kyseisen pyynnön eteenpäin LowLevelCANAPIlle(1.1), joka taas pyytää jokaista tietokoneeseen kytkettyä laitetta raportoimaan itsestään tietoja (1.1.1). Kun LowLevelCANAPI on kerännyt tiedot kaikista laitteista, tallentaa se tiedon itsellensä ja lähettää viitteen tietojen muistipaikkaan takaisin käyttöliittymälle, joka lukee laitteiden tiedot ja näyttää ne listattuna käyttäjälle. (Kuva 7)

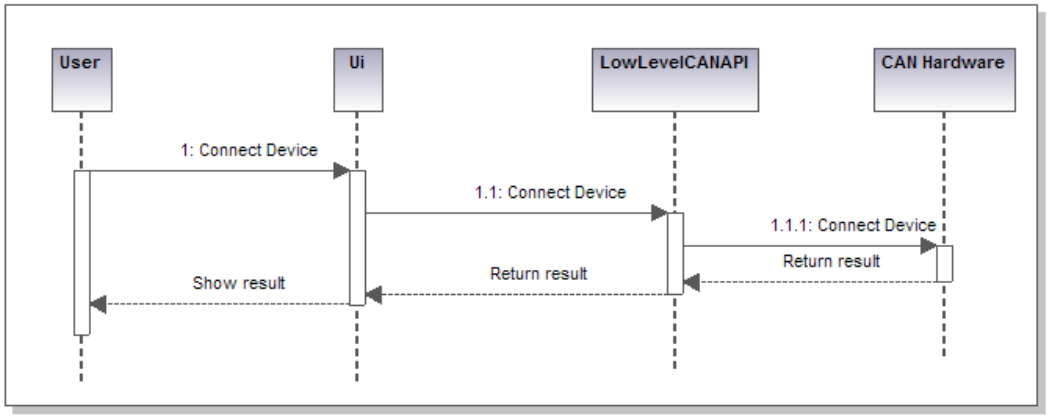


Kuva 7. Vaiheittainan kuvaus laitteiden hausta

2.4.5 Yhdistä väylälle

Käyttäjä käskee käyttöliittymän kautta yhdistää laitteen (tai laitteen kanavan) väylälle (1). Käyttöliittymä käskee LowLevelCANAPIa yhdistämään valittu laite (1.1). LowLevelCANAPI lähettää pyynnön CAN-laitteelle (1.1.1), joka suorittaa

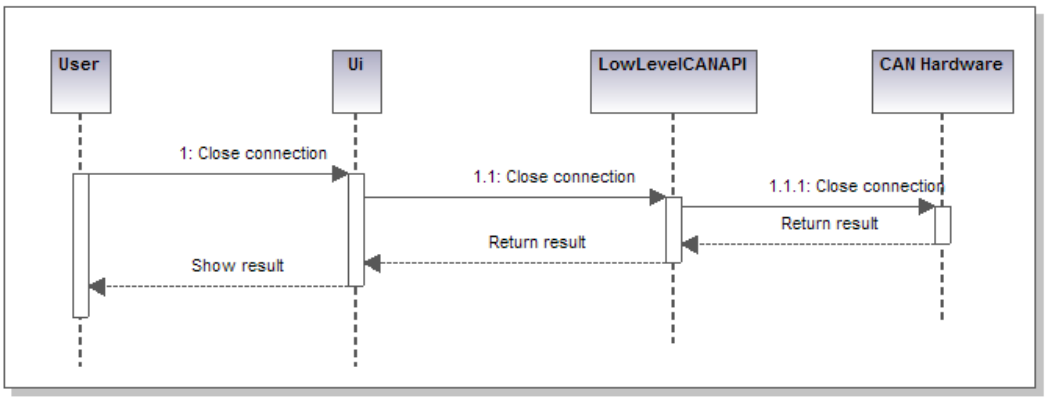
toiminnon ja raportoi takaisin tuloksesta. Tulos lähetetään takaisin käyttöliittymälle, joka näyttää tämän käyttäjälle. (Kuva 8)



Kuva 8. Vaiheittainen kuvaus väylälle yhdistämisestä

2.4.6 Sulje yhteys

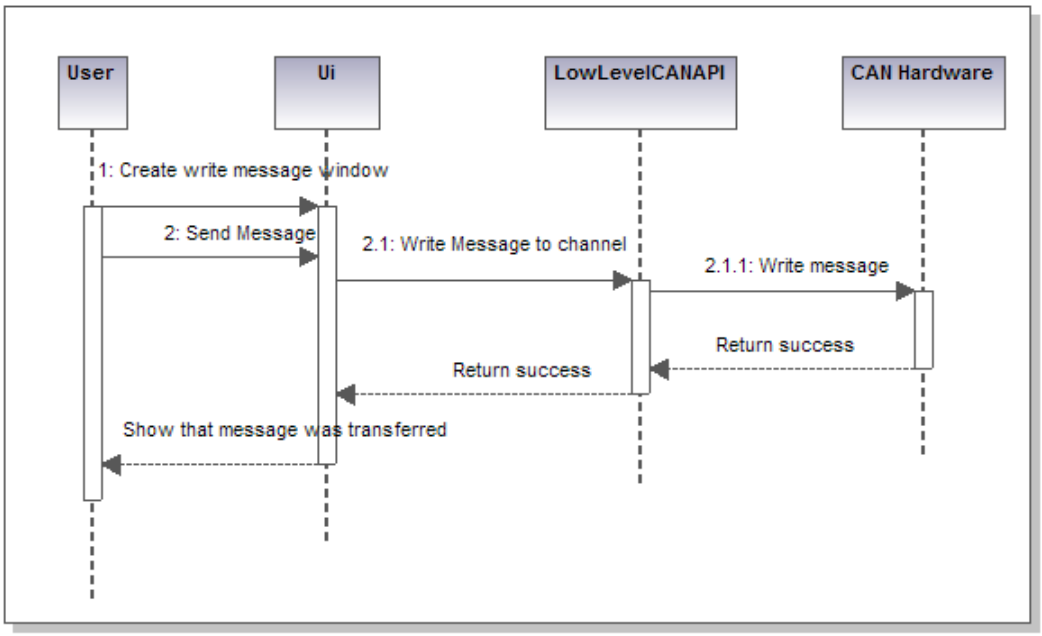
Käyttäjä käskee käyttöliittymän kautta sulkea laitteen (tai laitteen kanavan) yhteyden (1). Käyttöliittymä lähettää pyynnön LowLevelCANAPIlle (1.1), joka käskee laitetta sulkemaan yhteyden väylälle (1.1.1). Laite palauttaa tiedon operaation onnistumisesta LowLevelCANAPIlle, joka lähettää sen edelleen käyttöliittymälle, joka näyttää sen käyttäjälle. (Kuva 9)



Kuva 9. Vaiheittainen kuvaus väyläyhteyden sulkemisesta

2.4.7 Kirjoita viesti väylälle

Käyttäjä luo työpöydälle ikkunan viestin lähettämistä varten (1), ja konfiguroi ikkunaan viestin jonka haluaa lähettää. Käyttäjä käskee käyttöliittymän kautta haluavansa lähettää viestin (2). Käyttöliittymä välittää pyynnön viesteineen LowLevelCANAPIlle (2.1), joka pyytää CAN-laitetta lähettämään viestin (2.1.1). Laite palauttaa tiedon operaation onnistumisesta LowLevelCANAPIlle, joka välittää tiedon käyttöliittymälle. Käyttöliittymä kertoo käyttäjälle onnistuiko lähetys vai ei. (Kuva 10)



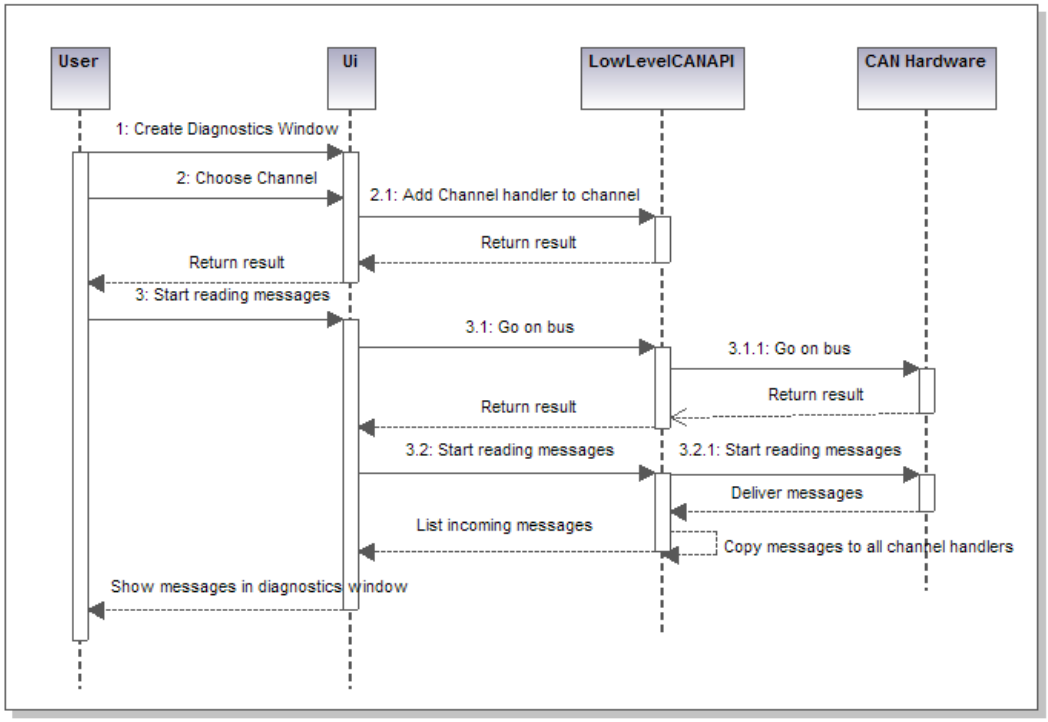
Kuva 10. Vaiheittainen kuvaus viestin kirjoittamisesta väylälle

2.4.8 Lue viestejä väylältä

Käyttäjä luo työpöydälle diagnostiikkaikkunan, johon haluaa ottaa vastaan viestejä (1). Ikkunan asetuksiin käyttäjä konfiguroi miltä kanavalta (tai kanavilta) käyttäjä haluaa ottaa vastaan viestejä (2). Kun kanava on valittu pyytää käyttöliittymä LowLevelCANAPIa luomaan uuden Channel handlerin, johon otetaan vastaan viestit (2.1). LowLevelCANAPI palauttaa tuloksen instanssin luonnista käyttöliittymälle, joka näyttää tuloksen käyttäjälle. Diagnostiikkikkunalla on oma säie, joka tarkastelee ChannelHandlerin viestipuskuria. Kun lukemattomia viestejä löytyy, ne luetaan ja sijoitetaan ikkunaan viestilistaan.

Käyttäjä pyytää käyttöliittymän kautta yhdistämään laite väylälle ja aloittamaan viestien vastaanoton (3). Käyttöliittymä käskee LowLevelCANAPIa yhdistämään laite väylälle (3.1). LowLevelCANAPI lähettää pyynnön laitteelle (3.1.1), ja palauttaa saamansa tuloksen käyttöliittymälle.

Jos virhettä ei esiinny, lähettää käyttöliittymä käskyn aloittaa viestien vastaanoton (3.2). LowLevelCANAPI käynnistää säikeen, joka lukee viestejä laitteelta sen mukaan kun niitä ilmestyy väylälle (3.2.1). Viestit välitetään LowlevelCANAPIsta käyttöliittymälle, joka latoo viestejä taulukkoon käyttäjän nähtäväksi. (Kuva 11)

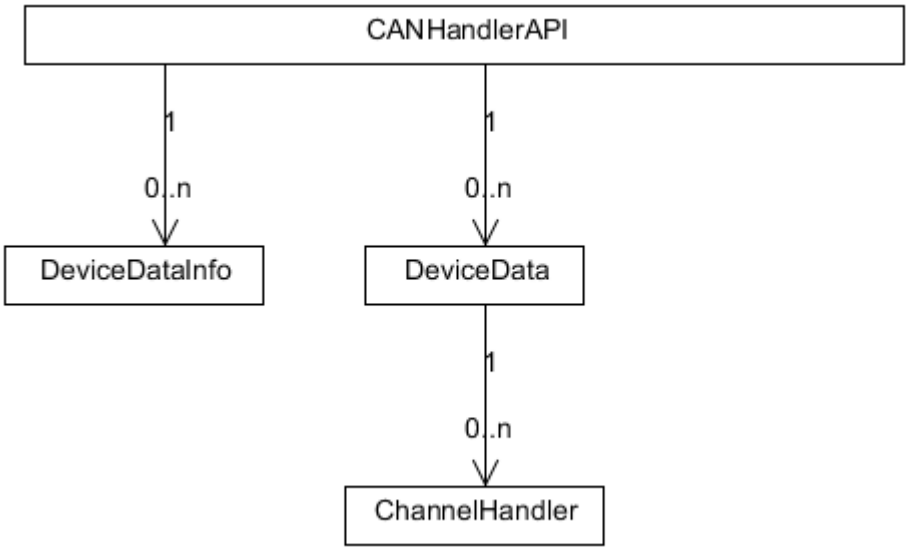


Kuva 11. Vaiheittainen kuvaus viestin lukemisesta väylältä

2.5 Sovelluksen luokat

LowLevelCANAPI sisältää kokonaisuudessaan 26 data- ja olioluokkaa, joten tässä esittellään vain kokonaisuuden kannalta tärkeimmät luokat. CANHandlerAPI on pääluokka, joka voi pitää sisällään DeviceData- ja DeviceDataInfo-luokkia, joiden määrä rajoittuu vain tietokoneeseen kytkettävien laitteiden määrästä. DeviceData voi taas pitää sisällään määräämättömän määrän ChannelHandler-olioita. DeviceDataInfo-luokka vastaa yhtä tietokoneeseen kytketyn laitteen kanavaa, kun taas DeviceData-luokka vastaa yhtä alustettua kanavaa. Erona näillä on että DeviceData-luokalla on enemmän muuttujia kuten

bOnBus ja muistipuskurit. (Kuva 12)



Kuva 12. Luokkien keskenäiset suhteet

2.5.1 CANHandlerAPI

Low Level CAN APIn pääluokka on CANHandlerAPI. Luokan kautta tehdään kaikki CAN-laitteita koskevat operaatiot kuten alustamisen, kirjoittamisen ja lukemisen. CANHandlerAPI-luokassa on myös tallessa suurin osa laitteiden ja viestien tiedoista. Alla olevassa taulukoissa (taulukko 5) kuvataaan CANHandlerAPIn metodit sekä niiden toiminnot.

Taulukko 5. CANHandlerAPI-funktionkuvaukset

Metodi	**Toiminto**
CANHandlerAPI();	luokan konstruktori
~CANHandlerAPI();	luokan destruktori
LookForAvailableDevices();	hakee tiedot kaikista kytketyistä CAN-laitteista
InitComm();	alustaa laitteen tai kanavan
DeinitComm();	poistaa laitteen käytöstä
GoOnBus();	yhdistää CAN-laitteen väylälle
GoOffBus();	CAN-laitteen väyläyhteyden sulkemista varten
GetBusLoad();	hakee väylän tämänhetkisen kuormituksen
GetStatus();	hakee väylän ja laitteen statustiedot
AddChannelHandler();	lisää laitteelle (tai laitteen kanavalle) ChannelHandlerin
RemoveChannelHandler();	poistaa laitteelta (tai laitteen kanavalta) ChannelHandlerin
WriteMessage();	kirjoittaa annetun viestin väylälle
ReadMessage();	lukee viestin annetulta ChannelHandlerilta
Thread_Run();	säie, joka hoitaa viestien monistamisen laitekohtaisilta puskureilta ChannelHandlerien puskureihin
MatchId();	noutaa viittauksen id:n perusteella

	annettuun laitteeseen.
MatchDriver();	palauttaa annettua laiteviittausta vastaavan DeviceData-instanssin
MatchHandler();	palauttaa viittauksen ChannelHandleriin annetun id:n perusteella
CalculateNextHandlerId();	laskee ja palauttaa seuraavan vapaan ChannelHandler id:n
DeviceExists();	kertoo onko annettua DeviceDataInfo-instanssia vastaava laite jo alustettu

Taulukko 6. CANHandlerAPI Attribuutit

Attribuutti	**Selitys**
ConnectedDevices*	vektori, jonne sijoitetaan kaikki tiedot PC:hen yhdistetyistä laitteista (tai niiden kanavista)
ConfiguredDevices*	vektori, jonne sijoitetaan kaikki tiedot alustetuista laitteista (tai niiden kanavista)
MediaDrivers *	vektori, jossa on viittaukset kaikkiin alustettuihin laitteisiin
Brunning	bool-arvo, jossa on tieto siitä onko CANHandlerAPI:n säie running-tilassa.
m_hThreadHandle	kahva CANhandlerAPIn säikeeseen
bKLibraryLoaded	bool-arvo, jossa on tieto siitä onko kvaserin kirjastot ladattu järjestelmään
bILibraryLoaded	bool-arvo, jossa on tieto siitä onko ixxatin kirjastot ladattu järjestelmään

2.5.2 DeviceData ja DeviceDataInfo

Muita tärkeitä luokkia ovat DeviceData ja DeviceDataInfo. DeviceDataInfo on datarakenne, jonne kerätään CANHandlerAPIn LookForAvailableDevices-metodissa CAN-laitteen tiedot, kuten laitteen nimi ja id. Tämän luokan instanssia käytetään kun laite alustetaan, jolloin luodaan DeviceData-instanssi vastaamaan alustettua laitetta. DeviceData-luokkaa taas käytetään laitteen operaatioita varten CANHandlerAPIn kautta. Laitteen operaatioita ei ole mahdollista suorittaa DeviceDataInfo-luokan instanssilla. CANHandlerAPIlla on kokoelmat joihin talletetaan kaikki DeviceData- ja DeviceDataInfo-luokat, sen mukaan kun niitä luodaan. Luokkien metodit ja attribuutit esitetään taulukoissa 7-10.

Taulukko 7. DeviceDataInfo funktionkuvaukset

Metodi	**Toiminto**
DeviceDataInfo();	luokan konstruktori
~DeviceDataInfo();	luokan destruktori

Taulukko 8. DeviceDataInfo Attribuutit

Attribuutti	**Selitys**
iUniqueId	uniikki laitekohtainen tunnistenumero
sDescription	tekstikuvaus laitteesta
sUiName	käyttöliittymässä käytettävä nimi laitteelle
iBaudrate;	muuttuja, johon asetetaan käyttäjän konfiguroima baudinopeus
iManufacturer;	numero, joka kertoo laitteen valmistajan
iChannels;	numero, joka kertoo kuinka monta kanavaa kyseisessä laitteessa on.
iChannel;	numero, joka kertoo mitä kanavaa kyseinen instanssi vastaa.

Taulukko 9. DeviceData funktionkuvaukset

Metodi	**Toiminto**
DeviceData();	luokan konstruktori. Toinen konstruktori ottaa parametrina DeviceDataInfo-instanssin, josta kopioidaan tieto valmiiksi tähän luokkaan.
~DeviceData();	luokan destruktori

Taulukko 10. DeviceData Attribuutit

Attribuutti	**Selitys**
deviceDriver *	viittaus luokkaa vastaavaan CAN-laiteluokkaan, jota käytetään kaikissa laitteen toiminnoissa
vchannelHandler	viittaus vektoriin, joka pitää sisällään kaikki kanavaan luodut ChannelHandlerit
iUniqueId	uniikki laitekohtainen tunnistenumero
sDescription	tekstikuvaus laitteesta
sUiName	käyttöliittymässä käytettävä nimi laitteelle
iBaudrate;	muuttuja, johon asetetaan käyttäjän konfiguroima baudinopeus
iManufacturer;	numero, joka kertoo laitteen valmistajan
iChannels;	numero, joka kertoo kuinka monta kanavaa kyseisessä laitteessa on
iChannel;	numero, joka kertoo mitä kanavaa kyseinen instanssi vastaa
bInitialized	bool-arvo, joka kertoo onko luokkaa vastaava CAN-laite alustettu
bOnBus	bool-arvo, joka kertoo onko laite väylällä

2.5.3 ChannelHandler

Jotta viestien vastaanotto toimisi sovelluksessa halutulla tavalla, pitää myös luoda ChannelHandler-instanssi, joka pääasiassa pitää sisällään puskurin vastaanotettuja viestejä varten. Yksi instanssi tästä luokasta vastaa yhtä viestipuskuria, johon kopioidaan kaikki kyseiseltä CAN-laitteelta tulleet viestit. Tämän luokan avulla voidaan ottaa vastaan saman kanavan viestejä, esimerkiksi kahteen ikkunaan. ChannelHandlerin metodit ja attribuutit esitetään taulukoissa 11 ja 12.

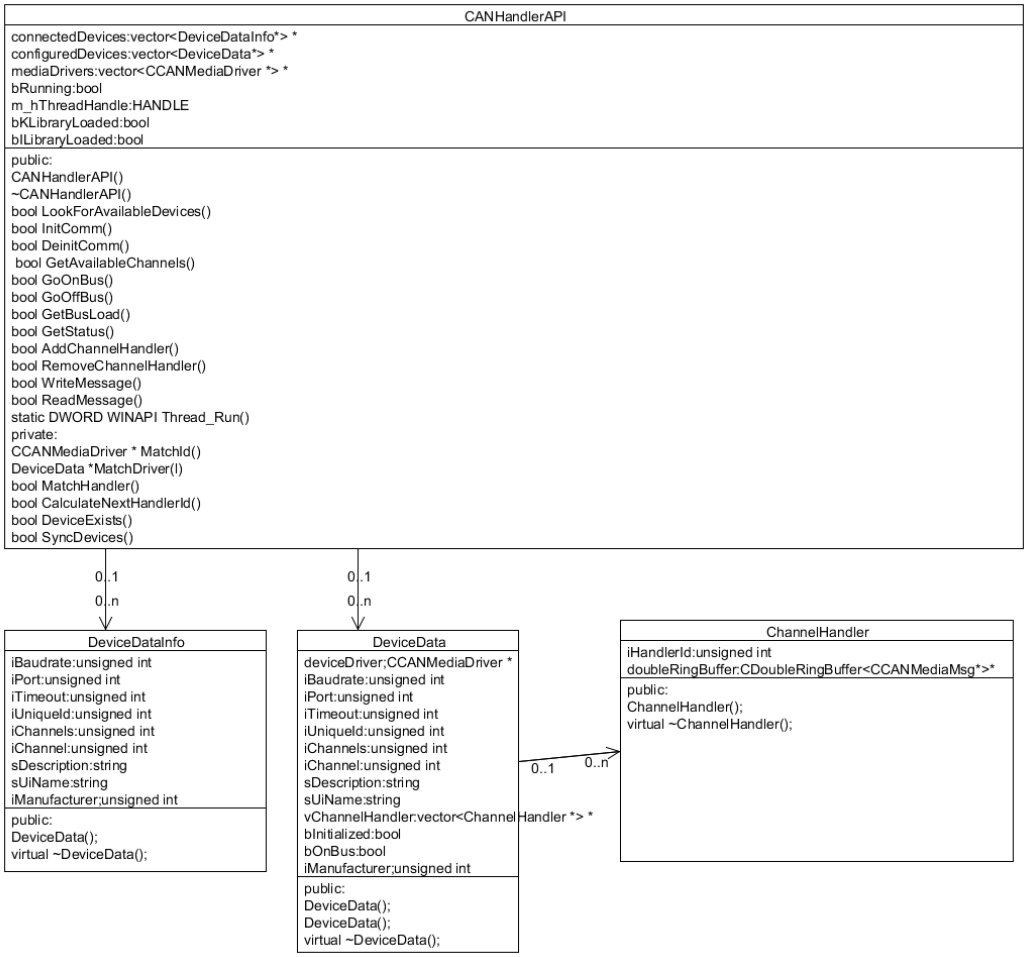
Taulukko 11. ChannelHandler funktionkuvaukset

Metodi	**Toiminto**
ChannelHandler();	luokan konstruktori
~ChannelHandler();	luokan destruktori

Taulukko 12. ChannelHandler Attribuutit

Attribuutti	**Selitys**
doubleRingBuffer *	rengaspuskuri jossa säilytetään vastaanotettuja viestejä
iHandlerId	uniikki Id-tunniste handlerille

CANHandlerAPI pitää sisällään kokoelmat, joissa sijaitsee kaikki DeviceData- ja DeviceDataInfo-luokat. Näitä luokkia ei sijaitse tästä irrallaan. Jokaisella DeviceData-oliolla on kokoelma ChannelHandler-olioita, jotka kuuluvat kyseiselle laitteelle. ChannelHandler-olioita ei myöskään voi sijaita DeviceDatan kokoelman ulkopuolella. Luokkien keskenäiset suhteet on esitetty kuvassa 13.

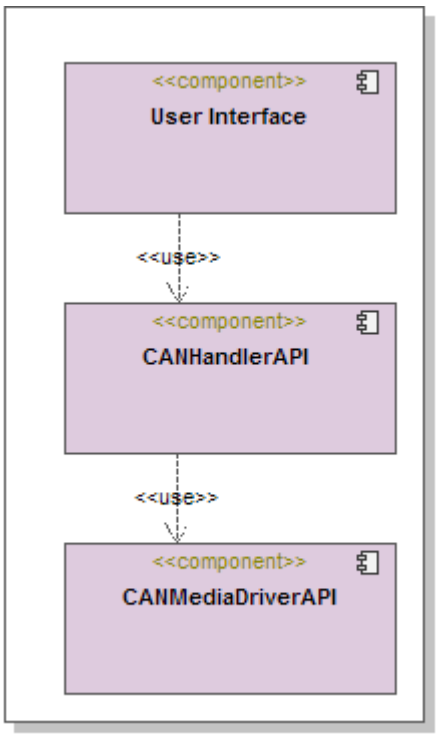


Kuva 13. Luokkien keskenäiset suhteet

2.6 Sovelluksen rakenne

Ohjelma jaettiin komponentteihin loogisten toimintatasojen perusteella. Alimpana sovelluksessa on CAN Mediadriver API, joka sisältää CAN-laitteiden kanssa keskustelevat luokat sekä läheisesti tähän liittyvät luokat kuten viesti- ja statusrakenteet. CAN Mediadriver API on yhteydessä CAN Handler API-komponenttiin, joka hoitaa sisäisesti mm. laitteiden alustuksen sekä organisoi toiminnalisuudet helposti käytettävään rajapintaan. Ylimpänä on käyttöliittymätaso, joka huolehtii graafisen näkymän muokkaamisesta sekä näyttämisestä käyttäjälle. Käyttöliittymätaso ei ole tietoinen kun CANHandlerAPI-komponentista, ja kaikki käyttöliittymän toiminnot jotka

vaativat CAN Mediadriver API-komponenttia tapahtuvat CANHandlerAPIn kautta. Sovelluksen rakenne on esitetty kuvassa 14.



Kuva 14 Sovelluksen rakenne

2.7 Arkkitehtuuri

Sovelluksen käyttöä varten tarvitaan aina PC. Tyypillisessä käyttötilanteessa PC:hen on kytketty USB- tai PCMCIA-väylään CAN-sovitin, joka taas on kytketty jollekin CAN-väylälle. CAN-sovitin on laite joka on kehitetty jotta PC:n saisi kytkettyä CAN-väylälle. Koska CAN-järjestelmää ei käytetä normaaleissa olosuhteissa koti-PC:n kanssa, ei PC:stä löydy valmiiksi rajapintaa tai laitteistoa CAN-väylän käyttöä varten. Lisälaitteita on olemassa monelta eri valmistajalta monenlaisin ominaisuuksin.

Väylällä voi olla yksittäinen laite tai väylä voi toimia kommunikaatioväylänä hyvinkin laajalle järjestelmälle. Tämän mukaista asetelmaa voi käyttää erilaiseen testaamiseen, vianetsintään, laitteiden uudellenkonfigurointiin, simulointiin tai

tutkimimustyöhön. Kuvassa 15 näkyy kuinka PC on kytketty CAN-laitteeseen ja CAN-laite on kytketty väylälle.

Kuva 15. Järjestelmän toiminta-arkkitehtuuri

3 KÄYTTÖLIITTYMÄN SUUNNITTELU

Koska vaatimuksena oli käyttäjän itse muokattava työpöytä, jäi tehtäväksi miettiä miten pilkotaan ohjelman ominaisuudet eri ikkunoihin työpöydälle. Pohdinnan jälkeen päädyttiin tekemään omat ikkunat viestin lukemiselle ja kirjoittamiselle. Omat ikkunat tehtiin myös väylän kuormittamiselle, triggerien määrittelemiselle, sekä nauhoitus-toiminnolle. Käyttöliittymään tehtiin valmiiksi käyttöliittymäkomponentit myös myöhemmin toteutettavia toimintoja varten.

3.1 Diagnostiikkaikkuna

Diagnostiikkaikkuna on viestien lukemista, väylän statuksennäyttöjä sekä näihin liittyviä toimintoja varten. Työkalupalkissa on näppäimet seuraavia toimintoja varten:

-lukemisen käynnistys ja keskeytys

-viestilistan tyhjennys

-viestien Suodinasetukset

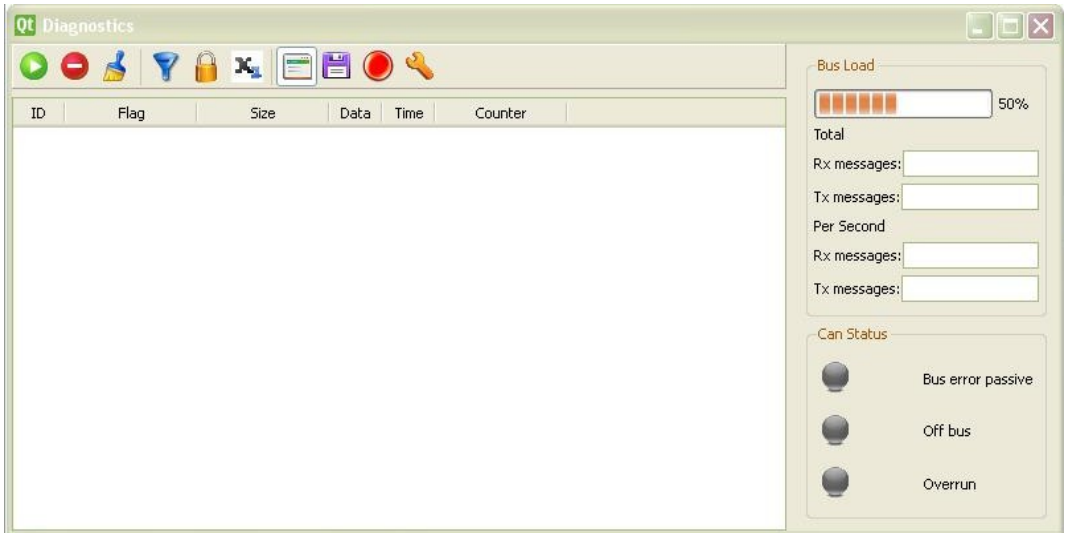
-locked mode on/off sekä heksadesimaali/desimaali-asetus

-oikean puolen statusnäkymän avaaminen sekä sulkeminen

-viestien tallennus tiedostoon

-ikkunan kanavien valinta

Kuvassa 16 on esitetty diagnostiikkaikkuna sovelluksessa.



Kuva 16. Viestien vastaanottoikkuna

3.2 Viestin lähetysikkuna

Ikkunassa voi konfiguroida lähetettävien viestien id-kentän, data-kentän sekä mahdolliset liput. Näiden lisäksi voi määrittää kuinka usein, ja kuinka monta kappaletta viestistä lähetetään. Viestien konfiguraation yhteydessä näkyy kuinka monta kyseistä viestiä on jo lähetetty, mutta tälle toiminnolle ei vielä ole toteutusta. (Kuva 17)

Kuva 17. Viestin lähetysikkuna

3.3 Kuormitusikkuna

Kuormitusikkuna on työkalu väylälle luotavaa kuormitusta varten tilanteessa, jossa esimerkiksi halutaan mitata jonkin järjestelmän suorituskykyä kun liikenne on kovin vilkasta. Työkalupalkissa on näppäimet kuormituksen käynnistykseen ja sulkemiseen. Näiden lisäksi löytyy ikkunasta erinäisiä kenttiä kuormituksen suuruden ja tyypin asettamista varten. On myös mahdollista määrittää minkälaisilla viesteillä väylää kuormitetaan. (Kuva 18)

Kuva 18: Kuormitusikkuna

4 TOTEUTUS

4.1.Työkalut

4.1.1 Tarvittavat työkalut

Laajan ohjelmistoprojektin toteuttamiseksi on valittava huolellisesti käytettävät työkalut ja työtavat, jotta projekti olisi vaivattomasti ylläpidettävissä, hallittavissa, ja kehitettävissä. Työkaluihin kuuluvat kehitysympäristö, kääntäjä ja tarvittavat ulkoiset oheislaitteet. Nämä valinnat vaikuttavat myös usein siihen mitä teknologioita kyseisessä projektissa voi soveltaa.

4.1.2 Kehitysympäristö

Kehitysympäristö valittiin siten, että seuraavat ominaisuudet toteutuivat :

-Kehitysympäristön pitää olla helposti muokattavissa omiin käyttötarkoituksiin.

-Kehitysympäristön pitää olla laajalti tuettu ja luotettava käyttää.

-Kehitysympäristöön tulee olla saatavilla tarvittavia liitännäisiä, jotta voidaan tehdä teknologialäheistä kehitystä.

-Koska projekti tulevaisuudessa luultavasti laajentuu Linux-pohjaisien järjestelmien käyttöön, tulee kehitysympäristön olla saatavilla myös Linux-puolelle.

Nämä vaatimukset huomioon ottaen valittiin kehitysympäristöksi Eclipse.

Eclipse on open source kehitysalusta. Open source on suomeksi avoin lähdekoodi, mikä tarkoittaa sitä, että kehitysalustaa ei ole tehty kaupalliseen käyttöön. Kuka tahansa voi tutkia sovelluksen rakennetta ja toteutusta sekä kehittää tätä edelleen.

Eclipse on alunperin 2001 IBM:än kehittämä projekti, joka muuttui 2004 itsenäiseksi ei-kaupalliseksi Eclipse Foundationiksi. Eclipsen projektit sisältävät

avoimen kehitysalustan koostuen laajoista ympäristöistä sekä työkaluista ohjelmiston kääntämiseen, implementoimiseen ja hallintaan. |1|

4.1.3 Kääntäjä

Kääntäjän kohdistuvat vaatimukset perustuvat lähelti kehitysympäristön vaatimuksiin. Näiden vaatimusten perusteella päädyttiin GNU compiler collection -kääntäjäpakettiin.

GNU compiler collection tai GCC, on osa GNU projektia, joka taas on yritys luoda ilmainen Unix-tyylinen käyttöjärjestelmä. Itse GCC on monelle alustalle ja monelle kielelle soveltuva kääntäjäkokoelma. GCC:tä ei myöskään ole tehty kaupalliseen tarkoitukseen. GCC on alunperin kehitetty Linux ympäristöön, mutta siitä on julkaistu versio myös Windows-käyttöön, MinGW. |5|

MinGW valittiin hyvien ominaisuuksien lisäksi kääntäjäkpaketiksi koska se on helposti otettavissa käyttöön Eclipse-ympäristössä.

4.1.4 Ulkoiset laitteet

Ulkoiset laitteet koostuvat itse projektin vaatimuksissa olevasta laitteistosta eli CAN-laitteisto, jota sovelluksen on tarkoitus tukea. Projektin vaatimuksiin kuuluu tuki Kvaserin ja IXXATin laitteistolle. Laitteisto tarkoittaa tässä USB ja PCMCIA-laitteita, joita käytetään kun on tarkoituksena päästä PC:llä käsiksi CAN-väylään. Lopputyön aikana on projektia testattu Kvaser leaflight sekä IXXAT USB-to-CAN II laitteilla. Näiden lisäksi on kehityksessä käytetty muutakin ulkoista laitteistoa. Nämä laitteet ovat asiakaskohtaisia, joten en voi tässä kertoa niistä enempää.

4.2 Teknologia

4.2.1 Toteutuskieli

Projektissa käytettin C++ -ohjelmointikieltä, koska käytettiin yrityksessä jo valmiiksi kehitettyjä C++ luokkia CAN-laitteiden käsittelyyn. Käytetty Qt-

käyttöliittymäkirjasto on C++ kirjasto, joten on luonnollista käyttää samaa kieltä koko projektissa. C++ soveltuu myös laiteläheisyytensä ja tehokkuutensa puolesta hyvin työhön.

4.2.2 Käyttöliittymä

Toteutettava graafinen käyttöliittymä on monipuolinen, joten siihen valittavan teknologian pitää olla laaja ja monikäyttöinen. Tulevaisuuden Linux-soveltuvuuden ja alustariippuvuuden takia valittiin Qt-käyttöliittymäkirjasto.

Qt-kirjaston on kehittänyt alunperin norjalainen Trolltech, jonka Nokia on ostanut vuonna 2008. Qt on monelle alustalle soveltuva kehitysympäristö sovellus- ja käyttöliittymäkehitykseen. Alustat, joille Nokia on julkaissut Qt:n ovat Linux/Unix, Mac OS X, Windows, sulautetut alustat (PDA, Smartphone jne.), Windows CE sekä S60. Tässä työssä valittiin käyttää vain Qt-käyttöliittymäkirjastoa, koska koettiin kehitysympäristö hankalakäyttöiseksi. |7|

Eclipseä varten löytyy Qt:n itsensä kehittämä liitännäinen, Qt Eclipse Integration, jonka avulla voi kehittää Qt-sovelluksia kätevästi Eclipsessä. Liitännäiseen kuuluu valmiiksi MinGW-kääntäjäkokoelma.

4.2.3 Laiteläheinen ohjelmointi

Ulkoisten laitteiden ohjelmointia varten laitevalmistajat tarjoavat omat valmiit ohjelmointirajapinnat. Näitä vasten on erääseen aiempaan projektiin toteutettu yhtenäiset rajapinnat eri valmistajien laitteisiin, joita tässä projektissa käyttettiin, tehden muutoksia tarpeen mukaan. Nämä rajapinnat on toteutettu C++-ohjelmointikielellä.

4.3 Viestin luku IXXAT-laitteelta

CAN-laitteet, joita käytetään tämän sovelluksen kanssa, tarjoavat ohjelmointirajapinnnan, jonka kautta laitteille voi itse kehittää ohjelmistoa. Tämä rajapinta on tietysti laitekohtainen, joten eri valmistajien laitteille piti toteuttaa

operaatiot erikseen.

Viestin lukuun IXXAT-laitteelta löytyy laitteen dokumentoinnista seuraava:

canChannelReadMessage

The function reads the next CAN message from the receive buffer of a message channel. The complete syntax of the function is:

```
HRESULT canChannelReadMessage(HANDLE hChannel, UINT32 dwMsTimeout, PCANMSG pCanMsg );
```

-*hChannel* on kahva kanavaan, josta halutaan lukea viesti

-*dwMsgTimeout* on unsigned integer-tyyppinen arvo, jolla määritellään kuinka kauan odotetaan viestin saapumista ennen kuin funktio palaa.

-*pCanMsg* on osoitin CANMSG tyyppiseen tietorakenteeseen, johon funktio kirjoittaa viestin tiedot mikäli viestejä otetaan vastaa sallitun ajan puitteissa.

Mikäli laitteelta löytyy viesti joka otetaan vastaan palauttaa funktio VCI_OK-vakion. Muussa tapauksessa palautuu virheeseen viittaava vakio.

CANMSG tietorakenne on kuvattu seuraavasti:

```
typedef struct{
    UINT32 dwTime;
    UINT32 dwMsgId;
    CANMSGINFO uMsgInfo;
    UINT8 abData[8];
} CANMSG, *PCANMSG;
```

-*dwTime* on aika “tickeinä” joka on laitteen oma aikayksikkö, joka vaihtelee laitteen ominaisuuksien mukaan. Laitteesta tulee hakea kaksi muuta arvoa, joiden avulla voidaan laskea yhden “tickin” resoluutio.

-*dwMsgId* on viestin id-kentän arvo.

-*uMsgInfo* on bittikenttä, joka kuvaa viestin tyyppiä.

-*abData* on taulukko, jossa on viestin data-osio.

Sovelluksessa viestin luku näyttää seuraavalta:

```
//Jos viestin luku onnistuu
if(pDriver->m_SDriverFkt.canChannelReadMessage(pDriver->m_hCANChannel1, 100,
    &Msg)==VCI_OK )
{
    switch(Msg.uMsgInfo.Bytes.bType)
    {
        //Jos viestin tyyppi on data tai error:
        case CAN_MSGTYPE_DATA:
        case CAN_MSGTYPE_ERROR:
        {
            //Luodaan uusi viestiolio
            pCanMsg = new CCANMediaMsg;
            //Asetetaan viestille timestamp
            DWORD dwTimeStamp = Msg.dwTime / pDriver->
                m_dwTimeRes + pDriver->m_dwTimeOverRun;
            pCanMsg->SetTimeStamp(dwTimeStamp);
```

Koodi toimii seuraavasti: Kysytään kanavalta m_hCANChannel viestiä canChannelReadMessage funktiolla. Timeoutiksi asetetaan 100 ms ja viesti talletetaan tietorakenteeseen Msg. Jos viesti otetaan vastaan onnistuneesti jatketaan laskemalla viestin timestamp. Aikaisemmin lasketaan CAN-laitteen väylälle yhdistävässä funktiossa resoluutio dwTimeRes kaavalla:

```
m_dwTimeRes=(DWORD)(1.0/(Caps.dwTscDivisor * 1000.0 / Caps.dwClockFreq));
```

Alkuperäisestä kaavasta poikkeava kerroin 1000 kaavassa tekee sen että tulos on millisekunteina. Resoluutio kertoo kuinka kauan aikaa on kahden tickin välissä. "Tickien" määrä jaetaan resoluutiolla niin saadaan kulunut aika. m_dwTimeOverRun on laskuri, joka pitää kirjaa siitä kuinka paljon lisäaikaa on oltava viestin aikalaskurin ympäripyörähtämisen vuoksi. Tämä lisäämällä saadaan laskettua aikaleima viestiin.

4.4 Viestin luku CANHandlerAPI-luokassa

CANHandlerAPI-luokassa on säie, joka hoitaa viestien siirtämisen Hardware-luokilta ChannelHandlereille, joista käyttöliittymä lukee vuorostaan viestit. Säie

käy yksinkertaisuudessaan kaikki CAN-laitteet vuoronperään läpi ja kopioi joka viestin jokaisesta laitteesta kaikille kyseisen laitteen ChannelHandler-olioille. Säie käsittelee vuorollaan eri valmistajien laitteet ja kopio aina kerralla yhdeltä laitteelta kaikki viestit ennenkuin siirtyy seuraavaan.

```
//Säie joka hoitaa viestien siirron Hardware-luokista

DWORD WINAPI CANHandlerAPI::Thread_Run(LPVOID lpParameter)
{
    CANHandlerAPI* pHandlerAPI = (CANHandlerAPI* )lpParameter;
    bool bMessagesLeft=false;

    //Pidetään säie käynnissä niin kauan kun bRunning=true

    while(pHandlerAPI->bRunning){

        //Luetaan viestit vuoronperään kaikista CAN-laitteista.

        for(unsigned int i=0;i<pHandlerAPI->mediaDrivers->size();i++)
        {

            bMessagesLeft=true;

            //Luetaan viestejä kunnes laitteelta ovat viestit loppu.

            while(bMessagesLeft)
            {

                //Jos laite on Kvaser

                if(typeid(*pHandlerAPI->mediaDrivers-
                        >at(i))==typeid(CKvaserMediaDriver))
                {
                    CKvaserMediaDriver *pKvaserDriver =
                            (CKvaserMediaDriver *)pHandlerAPI->mediaDrivers->at(i);
                    CCANMediaMsg *tmpMessage=new CCANMediaMsg();

                    //Lue viesti laitteelta.

                    bMessagesLeft=pKvaserDriver->ReadCanMsg(tmpMessage, 0);

                    //Jos säie halutaan pysäyttää, siirrytään silmukan loppuun.

                    if(!pHandlerAPI->bRunning)
                    {
                        bMessagesLeft=false;
                    }

                    //Jos saatiin viesti.

                    if(bMessagesLeft)
                    {
                        vector<ChannelHandler*> *pHandlers=pHandlerAPI-
```

```
                >MatchDriver(pKvaserDriver,0)->vChannelHandler;

            //Kopioidaan ja lisätään viesti kaikkien Channelhandlereitten
            //puskureihin.

            for(unsigned int j=0;j<pHandlers->size();j++)
            {

                CCANMediaMsg *cpyMsg=new CCANMediaMsg();
                cpyMsg->Copy(*tmpMessage);

                DWORD dwId=600;
                tmpMessage->GetCANID(dwId);

                pHandlers->at(j)->doubleRingBuffer->Add(cpyMsg);
            }

        }
        delete tmpMessage;
}

//Jos laite on ixxat.

else if(typeid(*pHandlerAPI->mediaDrivers-
        >at(i))==typeid(CIXXATMediaDriverV3))
{

    CIXXATMediaDriverV3 *pIXXATDriverV3 = (CIXXATMediaDriverV3
            *)pHandlerAPI->mediaDrivers->at(i);

    CCANMediaMsg *tmpMessage2=new CCANMediaMsg();

    //Luetaan viesti laitteelta.

    bMessagesLeft=pIXXATDriverV3->ReadCanMsg(tmpMessage2, 0);

    //Jos säie halutaan pysäyttää, siirrytään silmukan loppuun.

    if(!pHandlerAPI->bRunning)
    {
        bMessagesLeft=false;
    }

    //Jos saatiin viesti.

    if(bMessagesLeft)
    {
        vector<ChannelHandler*> *pHandlers=pHandlerAPI-
            >MatchDriver(pIXXATDriverV3,0)->vChannelHandler;

        //Kopioidaan ja lisätään viesti kaikkien
        //Channelhandlereitten puskureihin.

        for(unsigned int j=0;j<pHandlers->size();j++)
        {
```

```
                        CCANMediaMsg *cpyMsg2=new CCANMediaMsg();
                        cpyMsg2->Copy(*tmpMessage2);

                        DWORD dwId=0;
                        tmpMessage2->GetCANID(dwId);

                        pHandlers->at(j)->doubleRingBuffer->Add(cpyMsg2);
                    }
                }
                delete tmpMessage2;
            }
        }
    }
    Sleep(10);

}//Poistutaan säikeestä.

return 0;
}
```

4.5 Viestin kirjoittaminen väylälle

Viestin kirjoitus väylälle tapahtuu kutsumalla CANHandlerAPIn metodia WriteMessage. Metodi ottaa parametreina laitteen id:n, *iId*, osoittimen kirjoitettavaan viestiin, **pCCANMediaMsg* ja referenssin johon mahdollinen virheviesti kirjoitetaan, *&iErrorCode*. Metodi aloittaa selvittämällä minkätyyppisestä CAN-laitteesta on kyse ja tarkastaa tämän jälkeen, että laite on yhdistetty väylälle. Tämän jälkeen viesti kirjoitetaan ja tieto siitä onnistuiko kirjoitus vai ei palautetaan kutsuvalle ohjelmalle.

```
bool CANHandlerAPI::WriteMessage(unsigned int iId, CCANMediaMsg
    *pCCANMediaMsg, unsigned short int &iErrorCode)
{
    CCANMediaDriver *driver=MatchId(iId);
    bool bReturn=false;

    if(typeid(*driver)==typeid(CKvaserMediaDriver))
    {

        if(((CKvaserMediaDriver *)driver)->m_bBusOff)
        {
            iErrorCode=CCommunicationError::LLC_DEVICE_NOT_ON_BUS;
            return false;
        }

        bReturn=((CKvaserMediaDriver *)driver)->WriteCanMsg(pCCANMediaMsg);
    }
```

```
    if(typeid(*driver)==typeid(CIXXATMediaDriverV3))
    {

        if(((CIXXATMediaDriverV3 *)driver)->m_bBusOff)
        {
            iErrorCode=CCommunicationError::LLC_DEVICE_NOT_ON_BUS;
            return false;
        }

        bReturn=((CIXXATMediaDriverV3                                    *)driver)-
>WriteCanMsg(pCCANMediaMsg);
    }

    return bReturn;
}
```

4.6 Ikkunoiden luominen käyttöliittymässä

Qt:ssä ei monen muun käyttöliittymäkirjaston tapaan ole kuuntelijoita, jotka kuuntelevat tiettyjä tapahtumia ja reagoivat niiden perusteella. Sen sijaan käyttöliittymätoiminnot käyttävät signaaleja ja slotteja. Kun käyttöliittymässä tapahtuu jotain (esim. käyttäjä painaa painiketta), lähettää tämä tapahtuma signaalin. Slotit ovat funktioita, jotka voivat rekisteröityä näihin signaaleihin. Slotit ja signaalit eivät itsessään ole tietoisia toisistaan. Tämä on suurin ero kuuntelija-käytäntöön.

Käyttöliittymän alustuksesta:

```
newDiagWindowAct = new Qaction(QIcon("images/diagnostics.ico"),
      tr("New Diagnostics Window"), this);
newDiagWindowAct->setStatusTip(tr("Create new Diagnostics window"));
connect(newDiagWindowAct,              SIGNAL(triggered()),              this,
    SLOT(AddDiagWindow()));
```

newDiagWindowAct on QAction, eli toiminto, joka tässä vastaa uuden diagnostiikkaikkunan luomisesta. Ensimmäisellä rivillä luodaan uusi toiminto ja kerrotaan mitä ikonia ja nimeä tämä käyttää. Tämän jälkeen asetetaan toiminnolle statusteksti. Viimeisenä tapahtuu signaalin ja slotin yhdistäminen. Aina kun newDiagWindowAct antaa signaalin “triggered()” ajetaan slot “AddDiagWindow()”. Esimerkiksi painike voidaan luoda toiminnon avulla,

jolloin napin painaminen lähettää triggered signaalin. Yhden signaalin yhdistäminen moneen slottiin onnistuu myös ja vastaavasti monen eri signaalin yhdistäminen yhteen slottiin on mahdollista.

5 YHTEENVETO

5.1 Toteutus

Työssä toteutettiin sovellus, jonka avulla voi suorittaa CAN-väylään liittyvissä tilanteissa testausta, konfigurointia tai ongelmanratkaisua. Sovellus tehtiin C++-ohjelmointikielellä käyttäen Qt-käyttöliittymäkirjastoa sekä CAN-laitevalmistajien tarjoamia laiteläheisiä kirjastoja. Työssä toteutettiin kaikki pakolliset vaatimukset. Sovelluksesta tuli toimiva ja se täytti vaatimukset CAN-järjestelmien testaustyökaluna. Alemman tason CAN-kirjastossa toteutettiin yksinkertainen rajapinta ja kirjasto pitää kätevästi sisällään eri laitevalmistajien CAN-sovittimien hardware-käsittelyn.

Tämä projekti oli haasteellinen suuruusluokkansa vuoksi mutta hyvin opettavainen ohjelmistosuunnittelun ja -kehityksen suhteen. Suurimpia haasteita olivat näin laajan ohjelmiston suunnittelu ja toteuttaminen. Itse toteuttamisen aikana tuotti monen säikeen samanaikainen käsittely eniten päänvaivaa.

5.2 Jatkokehittelyn näkymät

Tulevaisuudessa tullaan sovellusta kehittämään eteenpäin ja siihen tullaan lisäämään toimintoja ja työkaluja, jotka tekevät siitä monipuolisemman ja voivat auttaa automatisoimaan sekä ketjuttamaan komentoja työn helpottamiseksi. Alatason CAN-kirjastoa tullaan myös tulevaisuudessa tiedettävästi käyttämään jo yhdessä projektissa ja toivottavasti monessa muussakin.

LÄHDELUETTELO

|1| About the Eclipse foundation [online] [viitattu 24.11.2009]. Saatavilla www-muodossa: <URL:http://www.eclipse.org/org/>.

|2| CANopen Device and Application profiles [online] [viitattu 23.11.2009]. Saatavilla www-muodossa: <URL:http://www.canopensolutions.com/english/about_canopen/profiles.shtml>.

|3| CAN in Automation [online]. [viitattu 9.11.2009]. Saatavilla www-muodossa: <URL:http://www.can-cia.de>.

|4| CAN Specification 2.0 [online] [viitattu 9.11.2009]. Saatavilla www-muodossa: <URL:http://www.can-cia.org/fileadmin/cia/specifications/CAN20A.pdf>.

|5| GCC, the GNU Compiler Collection [online] [viitattu 24.11.2009]. Saatavilla www-muodossa: <URL:http://gcc.gnu.org/>.

|6| Kuva [online] [viitattu 23.11.2009]. Saatavilla www-muodossa <URL:http://www.lightworkdesign.com/uploads/galleryimages/normal/galleryimage_r4IWA9HSzl.jpg>.

|7| Qt – A cross-platform application and UI framework [online] [viitattu 24.11.2009]. Saatavilla www-muodossa: <URL:http://qt.nokia.com/>.